Threshold Function Identification by Redundancy Removal and Comprehensive Weight Assignments

Chin-Heng Liu, Chia-Chun Lin, Yung-Chih Chen, Chia-Cheng Wu, Chun-Yao Wang, *Member, IEEE*, and Shigeru Yamashita, *Senior Member, IEEE*

***Abstract*—The identification of threshold function (TF), which determines whether a Boolean function can be represented by an linear threshold logic gate (LTG) or not, is a fundamental but important task in the theories of threshold logic. In this paper, we propose a more efficient and effective algorithm of TF identification by constructing the system of irredundant inequalities and adjusting the weight assignment comprehensively. This is the first non-ILP-based approach that is able to identify all the eight-input TFs. The experimental results demonstrated that the proposed approach is more effective than all the existing non-ILP-based approaches and the LTGs obtained by the proposed approach are optimal for near 100% cases. For TFs with 9–15 inputs, the proposed approach can identify 100 000 randomly generated TFs as well in a reasonable CPU time.**

***Index Terms*—Linear threshold logic gate (LTG), redundancy removal, threshold function (TF) identification, weight assignments.**

I. Introduction

THRESHOLD logic is a logic representation different from the traditional Boolean logic. It could represent a complex Boolean function using only one linear threshold logic gate (LTG). For example, a six-input Boolean function $f = x_1x_2 + x_1x_3x_4 + x_1x_3x_5 + x_1x_4x_5x_6 + x_2x_3x_4 + x_2x_3x_5x_6 + x_2x_4x_5x_6$ can be represented by one LTG as shown in Fig. 1. However, not every Boolean function can be represented by a single LTG. A Boolean function that can be represented by one LTG is called a threshold function (TF). For non-TFs, multiple LTGs are needed to realize them.

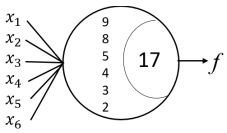


Fig. 1. Complex Boolean function $f = x_1x_2 + x_1x_3x_4 + x_1x_3x_5 + x_1x_4x_5x_6 + x_2x_3x_4 + x_2x_3x_5x_6 + x_2x_4x_5x_6$ represented by an LTG.

The fundamental study of threshold logic can be traced back to the 1960s. In 1961, an approach to enumerating TFs was first proposed [25]. Then, an approximation method for determining the weights and threshold value of an LTG was proposed as well [26]. Later, a linear programming method was proposed for determining whether a Boolean function is a TF or not [27]. Although many studies on threshold logic were conducted in early days, threshold logic did not make a great hit due to lack of efficient hardware realizations.

Recently, with the advances of emerging technologies such as quantum cellular automata [20], resonant tunneling diodes [2], resistant switching devices [9], spin-based devices [1], and single-electron-transistors [22], the implementations of LTGs become various and more easily [4]. Hence, threshold logic related studies, e.g., TF identification [10], [11], [19], synthesis or optimization of threshold logic networks [6], [13], [16], [29], timing analysis of threshold network [23], and approximate computing for threshold logic networks [14], [24], attract more attentions than ever before. Furthermore, LTGs are also strongly related to neural networks as well [12]. Actually, an LTG is a basic computation unit (neuron) of artificial neural networks [3].

The identification of TF, which determines whether a given Boolean function can be represented by an LTG or not, is a fundamental but important task in the theories of threshold logic. Furthermore, hardware cost can be reduced when we use only one LTG instead of many LTGs to realize a function. When a TF is recognized, the corresponding input weights and threshold value are determined as well. The approaches to this identification problem can be classified into two categories: 1) integer linear programming (ILP)-based approach [29] and 2) non-ILP-based approach [10], [11], [19]. The ILP-based approach [29] builds the system of inequalities from the function and exploits ILP solvers to obtain the weights and threshold value. However, the approach is not scalable due to its high complexity. That is, when the number of input variables of a function increases, the number

Manuscript received April 23, 2018; revised August 5, 2018; accepted September 18, 2018. Date of publication October 25, 2018; date of current version November 20, 2019. This work was supported in part by the Ministry of Science and Technology of Taiwan under Grant MOST 107-2221-E-155-046, Grant MOST 106-2221-E-007-111-MY3, and Grant MOST 103-2221-E-007-125-MY3, and in part by the Japan Society for the Promotion of Science under Grant JSPS/OF215/022 (S16042). This paper was recommended by Associate Editor S. Gao. *(Corresponding author: Chia-Cheng Wu.)*

C.-H. Liu, C.-C. Lin, C.-C. Wu, and C.-Y. Wang are with the Department of Computer Science, National Tsing Hua University, Hsinchu 30013, Taiwan (e-mail: posada2968@yahoo.com.tw; chiachunlin@gapp.nthu.edu.tw; orange392817@gmail.com; wcyao@cs.nthu.edu.tw).

Y.-C. Chen is with the Department of Computer Science and Engineering, Yuan Ze University, Taoyuan 32003, Taiwan (e-mail: ycchen.cse@saturn.yzu.edu.tw).

S. Yamashita is with the Department of Computer Science, Ritsumeikan University, Kyoto 525-8577, Japan (e-mail: ger@cs.ritsumei.ac.jp).

Digital Object Identifier 10.1109/TCAD.2018.2878181

of derived inequalities will grow exponentially such that the solvers cannot report the result within reasonable CPU time.

The non-ILP-based approaches [10], [11], [19], on the other hand, use heuristics to justify whether a Boolean function is a TF or not. Gowda *et al.* [10], [11] proposed an algorithm to decompose a Boolean function into subfunctions recursively. This process will be terminated when the subfunctions become AND, OR functions, or constant values 0, 1. Then, the algorithm merged these subfunctions and assigned weights and threshold value to the function. Although the algorithm was quite novel, the main drawback is that it can only identify all the TFs with three input variables. For the functions with 4–8 input variables, it cannot completely identify them.

In the state-of-the-art, Neutzling *et al.* [19] derived the system of inequalities from the irredundant sum-of-products (ISOPs) of a Boolean function. Then, the process of weight and threshold value assignment searches the weights and threshold value without violating the consistency of the system of inequalities. If an assignment is found, the function is identified as a TF; otherwise, the function is still undetermined. Although the method is able to identify all the TFs with up to six variables, it does not reach 100% for the functions with seven and eight variables. Furthermore, its derivation of system of inequalities contains many redundancies such that the succeeding assignment procedure is inefficient.

Thus, in this paper, we propose an improved non-ILP-based approach to TF identification. To the best of our knowledge, our approach is the first non-ILP approach that is able to identify all the eight-input TFs. The improvements of our approach come from two parts. First, the derivation of system of inequalities is more effective than the state-of-the-art [19] by constructing *irredundant* inequalities only. Hence, the number of inequalities that needs to be dealt with is decreased. Second, the procedure of weight assignment is more effective than the state-of-the-art by using a comprehensive heuristic.

Determining a function as a non-TF is easy, but confirming a function as a TF is quite challenging. Hence, to demonstrate the capability of our approach in TF identification, we construct *all* the NP-class TFs with 1–8 inputs as the benchmarks.

Furthermore, to the best of our knowledge, the set of all n-input TFs is not available publicly. Hence, we first generate the set of all n-input TFs for $n = 1$ to 8 for our experiments. However, it is not necessary to enumerate all the different n-input TFs for $n = 1$ to 8. Instead, only the NP-class TFs are sufficient. This is because the set of LTGs obtained from the NP-class TFs covers the LTGs obtained from all the TFs by negating and/or permuting the inputs. For example, given a three-input function $f = x_1 + x_2x_3$ in an NP-class, function $g = x_2 + x_1x_3$ also belongs to the same NP-class by permuting x_2 and x_1 in f. Also, function $h = x_1' + x_2x_3$ belongs to the same NP-class by negating x_1 in f.

According to the experimental results, our approach is able to identify all these benchmarks as TFs. Furthermore, the computed weights and threshold value of a TF are exactly the same as the optimal ones, which were obtained by ILP-based method [29], for near 100% cases.

The main contributions of this paper are threefold.

1) We propose an effective procedure to construct the system of *irredundant* inequalities for non-ILP-based approaches.
2) We propose a comprehensive heuristic to assign the optimal values to the weights and threshold value of TFs for 99.81% of all the eight-input NP-class TFs.
3) This is the first work that can identify all the eight-input NP-class TFs, 2 700 791 in total.

The rest of this paper is organized as follows. Section II introduces the background of threshold logic. Section III presents the proposed approach. Section IV shows the experimental results. Finally, Section V concludes this paper.

II. Preliminaries

This section introduces some fundamental concepts about threshold logic and terminologies related to the proposed approach.

A. Linear Threshold Gate

An LTG is a logic gate, which can represent a TF with n binary inputs and a binary output. Each input x_i has its own weight w_i, and the gate has a threshold value T. An LTG can be represented by a *weight-threshold vector* $<w_1, w_2, \ldots, w_i, \ldots, w_n; T>$. The mechanism of output evaluation is that if the weighted summation of an input pattern is greater than or equal to T, the output f is evaluated to 1; otherwise, 0, as shown in the following equation:

$$f(x_1, x_2, \ldots, x_n) = \begin{cases} 1, & \text{if } \sum_{i=1}^{n} x_i w_i \geq T \\ 0, & \text{if } \sum_{i=1}^{n} x_i w_i < T \end{cases} \quad (1)$$

where the parameters w_i and T are integers. For example, an LTG $<9, 8, 5, 4, 3, 2; 17>$ as shown in Fig. 1 has the output of 1 under the input pattern 110001, since $9+8+2 = 19 \geq 17$. Similarly, the LTG has the output of 0 under the input pattern 011001.

B. Threshold Function

A TF is a subset of Boolean functions that can be represented by an LTG. All the TFs are unate functions. That is, a binate function cannot be a TF. However, not every unate function is a TF. For example, a unate function $f = x_1x_2 + x_3x_4$ is not a TF.

C. Irredundant Sum-of-Products

Sum-of-products (SOP) is an expression to represent a Boolean function. When a Boolean function is expressed by ORing product terms only, where a product term is ANDed by literals, it is called an SOP expression. ISOP is an SOP expression that no product term can be removed without changing the functionality. The ISOP expression of a function is used as input for generating system of irredundant inequalities in the state-of-the-art and our approach. Although the proposed TF identification algorithm adopts the ISOP expression as input, there are some previous works focusing on the ISOP expression generation [8], [17], [21], [28]. For example, Minato [17]

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Fig. 2. Truth table of $f = x_1 + x_2x_3$.

Greater side		*Lesser side*
$w_1 + w_2$		$w_3 + w_4 + w_5 + w_6$
$w_1 + w_3$		$w_2 + w_5 + w_6$
$w_1 + w_4$	$\geq$ T $>$	$w_1 + w_6$
$w_2 + w_3$		$w_1 + w_5$
$w_2 + w_4$		---
$w_1 + w_5 + w_6$		---

Fig. 3. List of weighted summations for the on-set and off-set cubes of f [19].

proposed a BDD-based algorithm to generate irredundant covers. We can easily reimplement these methods to get the ISOP expression of any Boolean function.

D. Modified Chow's Parameter

An input of a TF corresponds to a weight in its LTG representation. Hence, there exists a weight ordering among all inputs of a TF.

Muroga [18] proposed the *modified Chow's parameter*, which is used to determine the variable ordering of a TF f. The formula of modified Chow's parameter is listed in the following equation:

$$q_i(f) = 2 \times p_i(f) - p_0(f) \tag{2}$$

where $q_i(f)$ represents the parameter of the ith input variable of TF f, $p_i(f)$ is the number of minterms in the on-set for which $x_i = 1$, and $p_0(f)$ is the number of minterms in the on-set of this function. By sorting the parameters $q_i(f)$ in the descending order for $i = 1, \ldots, n$, we can obtain the corresponding variable ordering of an n-input TF.

For example, given $f = x_1 + x_2x_3$, its truth table is shown in Fig. 2. The number of minterms in the on-set is $p_0(f) = 5$. Four of them are with $x_1 = 1$, and three of them are with $x_2 = 1$ or $x_3 = 1$. Therefore, the values of $p_1(f), p_2(f)$, and $p_3(f)$ are 4, 3, and 3, respectively. By (2), we can obtain $q_1(f) = 3$, $q_2(f) = 1$, and $q_3(f) = 1$. Hence, the variable ordering $w_1 > w_2 = w_3$ is determined.

E. Critical Effect Vector

Critical effect vector (CEV) is an assignment for an LTG such that the output changes from 1 to 0 when any one of its inputs in this assignment changes from 1 to 0. For example, 110 and 101 are the only CEVs for the LTG **<2, 1, 1; 3>**. CEVs can be derived by the algorithm proposed in [15].

III. Threshold Function Identification

In this section, we first review the state-of-the-art [19]. Then, we present the proposed approach in detail.

A. Review of the State-of-the-Art

The best way to review the state-of-the-art [19] is to use an example to demonstrate its identification steps for TFs. Given $f = x_1x_2 + x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_1x_5x_6$ [19], its on-set $= \{x_1x_2, x_1x_3, x_1x_4, x_2x_3, x_2x_4, x_1x_5x_6\}$. First, the algorithm checks whether the function is unate or not. If the function is not unate, the function is not a TF. Since the polarity of each literal in f is positive, f is unate.

Then, the algorithm computes a variable weight ordering (VWO) of the function, which is a vector like the modified Chow's parameter for determining the variable ordering. The resultant variable ordering of function f is $x_1 > x_2 > x_3 = x_4 > x_5 = x_6$.

The next step is to generate the system of inequalities, which consists of three parts.

1) Generation of inequalities from the ISOP.
2) Construction of the system of inequalities.
3) Simplification of the system of inequalities.

To generate the inequalities, the complemented function $f' = x_1'x_2' + x_1'x_3'x_4' + x_2'x_3'x_4'x_5' + x_2'x_3'x_4'x_6'$ is needed. Hence, the off-set $= \{x_1'x_2', x_1'x_3'x_4', x_2'x_3'x_4'x_5', x_2'x_3'x_4'x_6'\}$ is obtained. Since there may exist redundant cubes in the on-set and off-set during the generation of inequalities from the ISOP, it is not necessary to compare all the minterms/cubes in the on-set with all the minterms/cubes in the off-set. That is, for a minterm/cube i in the on-set whose weighted summation is *greater than* or equal to that of a minterm/cube j in the on-set, the minterm/cube i is a redundant minterm/cube. Similarly, for a minterm/cube i in the off-set whose weighted summation is *less than* or equal to that of a minterm/cube j in the off-set, the minterm/cube i is redundant as well. For example, if cubes $x_1x_2x_3$ and $x_1x_2x_3'$ are both in the on-set under the variable ordering $x_1 > x_2 > x_3$, the weighted summation of cube $x_1x_2x_3$, i.e., $w_1 + w_2 + w_3$ is greater than that of cube $x_1x_2x_3'$, i.e., $w_1 + w_2$. Hence, cube $x_1x_2x_3$ is a redundant cube and can be removed. In summary, for this example, to get the least cubes in the on-set, the don't care bits of one cube are set to 0, i.e., cube x_1x_2 is corresponding to $w_1 + w_2$. Similarly, to get the greatest cubes in the off-set, the don't care bits of one cube are set to 1, i.e., cube $x_1'x_2'$ is corresponding to $w_3 + w_4 + w_5 + w_6$. Thus, the inequalities are generated as shown in Fig. 3, where the greater side is for the on-set cubes, the lesser side is for the off-set cubes, and w_i represents the weight of x_i. Next, the algorithm pairs the weighted summation in the greater side with that in the lesser side as shown in Fig. 3 to construct the system of inequalities with $6 \times 4 = 24$ inequalities as shown in Fig. 4.

Since Fig. 4 may still contain redundant inequalities, the algorithm simplifies the system of inequalities by removing redundant inequalities. In this example, the algorithm uses the same symbol to replace the same weight, i.e., $A = w_1, B = w_2, C = w_3 = w_4$, and $D = w_5 = w_6$. Hence, the identical symbols in both sides among all the inequalities can be removed. For example, the inequality #2

#	Inequality	#	Inequality
1	$w_1+w_2>w_3+w_4+w_5+w_6$	13	$w_2+w_3>w_3+w_4+w_5+w_6$
2	$w_1+w_2>w_2+w_5+w_6$	14	$w_2+w_3>w_2+w_5+w_6$
3	$w_1+w_2>w_1+w_6$	15	$w_2+w_3>w_1+w_6$
4	$w_1+w_2>w_1+w_5$	16	$w_2+w_3>w_1+w_5$
5	$w_1+w_3>w_3+w_4+w_5+w_6$	17	$w_2+w_4>w_3+w_4+w_5+w_6$
6	$w_1+w_3>w_2+w_5+w_6$	18	$w_2+w_4>w_2+w_5+w_6$
7	$w_1+w_3>w_1+w_6$	19	$w_2+w_4>w_1+w_6$
8	$w_1+w_3>w_1+w_5$	20	$w_2+w_4>w_1+w_5$
9	$w_1+w_4>w_3+w_4+w_5+w_6$	21	$w_1+w_5+w_6>w_3+w_4+w_5+w_6$
10	$w_1+w_4>w_2+w_5+w_6$	22	$w_1+w_5+w_6>w_2+w_5+w_6$
11	$w_1+w_4>w_1+w_6$	23	$w_1+w_5+w_6>w_1+w_6$
12	$w_1+w_4>w_1+w_5$	24	$w_1+w_5+w_6>w_1+w_5$

Fig. 4. Original system of inequalities of f [19].

#	Inequality	#	Inequality
1	A+B > C+C+D+D	13	B > C+D+D
2	A > D+D	14	C > D+D
3	B > D	15	B+C > A+D
4	B > D	16	B+C > A+D
5	A > C+D+D	17	B > C+D+D
6	A+C > B+D+D	18	C > D+D
7	C > D	19	B+C > A+D
8	C > D	20	B+C > A+D
9	A > C+D+D	21	A > C+C
10	A+C > B+D+D	22	A > B
11	C > D	23	D > 0
12	C > D	24	D > 0

Fig. 5. System of inequalities of f with reformatted symbols [19].

#	Inequality		
1	A+B	>	C+C+D+D
2	A	>	D+D
5	A	>	C+D+D
6	A+C	>	B+D+D
13	B	>	C+D+D
14	C	>	D+D
15	B+C	>	A+D
21	A	>	C+C

Fig. 6. Final system of simplified inequalities of f [19].

"$w_1 + w_2 > w_2 + w_5 + w_6$" in Fig. 4 can be transformed into "$A + B > B + D + D$," and then be simplified as "$A > D + D$" by removing the symbol B. As a result, the system of inequalities in Fig. 4 can be reformatted as Fig. 5. The redundant inequalities in Fig. 5, which are either repeated inequalities or compatible with the VWO, are eliminated as well. For example, the inequalities #3 and #4 are repeated; the inequality #22 is compatible with the VWO $A > B > C > D$. After eliminating the redundant inequalities, the final system of inequalities is as shown in Fig. 6. The number of remaining inequalities is reduced from 24 to 8.

The last step of TF identification is the weight assignment and threshold value computation. The algorithm heuristically determines a weight assignment (7, 6, 3, 3, 1, 1) to the variables of $(x_1, x_2, x_3, x_4, x_5, x_6)$.[1] The threshold value is computed as the least weighted summation of the cube in the on-set. In this example, the weighted summations of the on-set cubes are 13, 10, 10, 9, 9, and 9, respectively. Hence, 9 is chosen as the threshold value. At this moment, the weights and threshold value of the function f are found successfully, and f is identified as a TF.

[1]We will use another complex example to demonstrate the process of weight assignments.

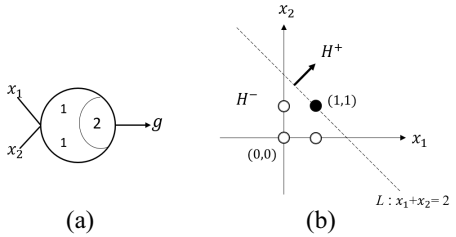


Fig. 7. (a) LTG of $g = x_1x_2$ [16]. (b) Hyperplane and hyperspaces of an AND gate [16].

Greater side		*Lesser side*
w_2+w_3		$w_3+w_4+w_5+w_6$
$w_1+w_5+w_6$	$\geq T >$	$w_2+w_5+w_6$
		w_1+w_5

Fig. 8. List of irredundant weighted summations in the greater side and lesser side of f.

#	Inequality		
1	w_2+w_3	>	$w_3+w_4+w_5+w_6$
2	w_2+w_3	>	$w_2+w_5+w_6$
3	w_2+w_3	>	w_1+w_5
4	$w_1+w_5+w_6$	>	$w_3+w_4+w_5+w_6$
5	$w_1+w_5+w_6$	>	$w_2+w_5+w_6$
6	$w_1+w_5+w_6$	>	w_1+w_5

Fig. 9. System of inequalities of f.

B. Generation of the System of Irredundant Inequalities

In Section III-A, we used an example to demonstrate the algorithm of the state-of-the-art [19]. Although the algorithm simplified the system of inequalities significantly, we still observed that the system of inequalities contained redundancy.

Let us first explain the reason behind this redundant inequality generation. An important characteristic of TF is *linear separability* [16], which means that there exists a hyperplane separating the on-set cubes and off-set cubes in two hyperspaces for a TF. For example in Fig. 7, given $g = x_1x_2$ and its LTG, there exists a hyperplane $L : \sum_{i=1}^{2} x_i w_i = T$ separating cube x_1x_2 in H^+ from cubes $x_1'x_2, x_1x_2', x_1'x_2'$ in H^-. Since the weighted summations of the least cubes in the on-set are always larger than that of the greatest cubes in the off-set, a naive way to generating the system of inequalities is to pair up all the weighted summations with respect to these cubes exhaustively as did in the state-of-the-art [19]. However, some weighted summations of the greater side or the lesser side are *redundant* when considering the computed VWO.

Thus, in this paper, we propose two redundancy removal steps to obtain the system of irredundant inequalities efficiently as follows.

#	Inequality		
1	w_2+w_3	>	$w_3+w_4+w_5+w_6$
2	w_2+w_3	>	$w_2+w_5+w_6$
3	w_2+w_3	>	w_1+w_5
4	$w_1+w_5+w_6$	>	$w_3+w_4+w_5+w_6$

Fig. 10. System of irredundant inequalities of f.

Greater side		*Lesser side*
$w_1+w_2+w_7$		$w_3+w_4+w_6+w_7$
$w_1+w_3+w_6$		$w_1+w_5+w_6$
$w_1+w_5+w_6+w_7$	$\geq T >$	w_1+w_2
$w_3+w_4+w_5$		$w_1+w_3+w_7$

Fig. 11. List of irredundant weighted summations in the greater side and lesser side of s.

#	Inequality		
1	w_1+w_2	>	$w_3+w_4+w_6$
2	w_1+w_7	>	w_5+w_6
3	w_1	>	w_4+w_7
4	w_3+w_6	>	w_2
5	w_1+w_5	>	w_3+w_4
6	w_5+w_6	>	w_3
7	$w_5+w_6+w_7$	>	w_2
8	w_5	>	w_6+w_7
9	w_4+w_5	>	w_1+w_7
10	w_3+w_4	>	w_1+w_6
11	$w_3+w_4+w_5$	>	w_1+w_2

Fig. 12. System of irredundant inequalities of s.

1) Redundant Weighted Summation Removal: For the same function $f = x_1x_2 + x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_1x_5x_6$ in Section III-A, Fig. 3 lists six weighted summations in the greater side and four weighted summations in the lesser side.[2] However, according to the computed VWO $x_1 > x_2 > x_3 = x_4 > x_5 = x_6$ by using the modified Chow's parameter or VWO, we can further remove some redundant weighted summations in the greater side or lesser side. For example, since $w_1 + w_2$ is larger than $w_2 + w_3$ in the greater side under the variable ordering $x_1 > x_2 > x_3 = x_4 > x_5 = x_6$, $w_1 + w_2$ is redundant and can be removed. As a result, Fig. 3 can be updated as Fig. 8, where two weighted summations and three weighted summations are left in the greater side and lesser side, respectively.

2) Redundant Inequality Removal: In this step, to construct the system of irredundant inequalities, the algorithm pairs the irredundant weighted summations in the greater side with that in the lesser side as shown in Fig. 9. However, there might still exist redundant inequalities after the pairing procedure. For an inequality i, when the weighted summation in the greater side is larger than that in the lesser side under the computed variable ordering, the inequality i is definitely satisfiable and does not need to be further considered. For example, given a weighted summation w_1+w_2 in the greater side and w_3+w_4 in the lesser side under the weight ordering $w_1 > w_2 = w_3 > w_4$, the inequality "$w_1 + w_2 > w_3 + w_4$" is definitely satisfiable. Hence, it is a redundant inequality and does not need to be constructed during the pairing procedure.

For this demonstrative example $f = x_1x_2 + x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_1x_5x_6$, the weighted summation of $w_1 + w_5 + w_6$ is larger than that of $w_2 + w_5 + w_6$. Hence, the inequalities #5 "$w_1 + w_5 + w_6 > w_2 + w_5 + w_6$" in Fig. 9 is a redundant inequality and does not need to be constructed. After conducting this redundant inequality removal step, the final system of irredundant inequalities is constructed as shown in Fig. 10. Compared with the state-of-the-art on this example, the number of remaining inequalities is reduced from 8 to 4, or 50% reduction. This reduction can lead the succeeding procedure, i.e., weight assignment, to be more efficient.

Note that we could also identify a function as a non-TF in this step of redundant inequality removal. That is, when a resultant inequality in the final system of irredundant inequalities violates linear separability, the function can be identified as a non-TF, and the procedure of TF identification is terminated. For example, given $h = x_1x_2 + x_3x_4$ under the variable ordering $x_1 = x_2 = x_3 = x_4$, where an irredundant cube x_1x_2 is in the on-set, and an irredundant cube x_1x_3 is in the off-set. Then, we can obtain an inequality $w_1 + w_2 > w_1 + w_3$. However, according to the variable ordering, the weighted summation of cube x_1x_2 should be equal to that of cube x_1x_3. Thus, h is identified as a non-TF.

[2]These weighted summations are corresponding to the cubes in the on-set and off-set.

C. Weight Assignment Procedure

In this section, we use a more complex example to demonstrate the procedure of weight assignment in the state-of-the-art [19]. Then, we present our new procedure of weight assignment in detail.

1) Review of the State-of-the-Art [19]: The example we use here is a TF that the state-of-the-art cannot identify successfully. Given a seven-input function $s = x_1x_2x_3 + x_1x_2x_4 + x_1x_2x_5 + x_1x_2x_6 + x_1x_2x_7 + x_1x_3x_4 + x_1x_3x_5 + x_1x_3x_6 + x_1x_4x_5 + x_1x_4x_6 + x_1x_5x_6x_7 + x_2x_3x_4 + x_2x_3x_5 + x_2x_3x_6 + x_2x_4x_5 + x_2x_4x_6 + x_2x_5x_6x_7 + x_3x_4x_5$, the irredundant weighted summations in the greater side and lesser side have been computed as Fig. 11. Its final system of irredundant inequalities has been derived as well by using our method under the variable ordering $x_1 = x_2 > x_3 = x_4 > x_5 > x_6 > x_7$ as shown in Fig. 12.[3] By replacing the same weight w_i with the same symbol, i.e., $A = w_1 = w_2, B = w_3 = w_4, C = w_5, D = w_6, E = w_7$, the system of inequalities is reformatted as Fig. 13.

The objective of this assignment procedure is to assign a least weight to each variable such that *all* the inequalities are satisfiable. If an assignment can be found, the function is a TF; otherwise the function is identified as an *undetermined* function. Hence, the weight assignment algorithm first assigns a least positive integer to each weight satisfying the variable ordering $A > B > C > D > E$, i.e., $A = 5, B = 4, C = 3, D = 2$, and $E = 1$. This assignment is called an *initial weight assignment*. Then, the system of inequalities is obtained under this assignment as shown in

[3]We focus on the weight assignment procedure in this example.

#	Inequality		
1	A+A	>	B+B+D
2	A+E	>	C+D
3	A	>	B+E
4	B+D	>	A
5	A+C	>	B+B
6	C+D	>	B
7	C+D+E	>	A
8	C	>	D+E
9	B+C	>	A+E
10	B+B	>	A+D
11	B+B+C	>	A+A

Fig. 13. System of irredundant inequalities of s with reformatted symbols.

#	Inequality			
1	**A+A = 10**	**>**	**B+B+D = 10**	**(false)**
2	A+E = 6	>	C+D = 5	
3	**A = 5**	**>**	**B+E = 5**	**(false)**
4	B+D = 6	>	A = 5	
5	**A+C = 8**	**>**	**B+B = 8**	**(false)**
6	C+D = 5	>	B = 4	
7	C+D+E = 6	>	A = 5	
8	**C = 3**	**>**	**D+E = 3**	**(false)**
9	B+C =7	>	A+E = 6	
10	B+B = 8	>	A+D = 7	
11	B+B+C = 11	>	A+A = 10	

Fig. 14. Weighted summation of the system of irredundant inequalities of s under the initial weight assignment $A = 5, B = 4, C = 3, D = 2$, and $E = 1$.

#	Inequality			
1	**A+A = 12**	**>**	**B+B+D = 12**	**(false)**
2	A+E = 7	>	C+D = 6	
3	**A = 6**	**>**	**B+E = 6**	**(false)**
4	B+D = 7	>	A = 6	
5	**A+C = 10**	**>**	**B+B = 10**	**(false)**
6	C+D = 6	>	B = 5	
7	C+D+E = 7	>	A = 6	
8	C = 4	>	D+E = 3	
9	B+C = 9	>	A+E = 7	
10	B+B = 10	>	A+D = 8	
11	B+B+C = 14	>	A+A = 12	

(a)

#	Inequality			
1	A+A = 14	>	B+B+D = 12	
2	A+E = 8	>	C+D = 6	
3	A = 7	>	B+E = 6	
4	**B+D = 7**	**>**	**A = 7**	**(false)**
5	A+C = 11	>	B+B = 10	
6	C+D = 6	>	B = 5	
7	**C+D+E = 7**	**>**	**A = 7**	**(false)**
8	C = 4	>	D+E = 3	
9	B+C = 9	>	A+E = 8	
10	B+B = 10	>	A+D = 9	
11	**B+B+C = 14**	**>**	**A+A = 14**	**(false)**

(b)

Fig. 15. Weighted summation of the system of irredundant inequalities of s under the (a) weight assignment $(6, 5, 4, 2, 1)$ and (b) weight assignment $(7, 5, 4, 2, 1)$.

Fig. 14, where some inequalities are not held. These inequalities are called *false inequalities*. Next, it adjusts the weights from the smallest weight in an ascending order. However, if the greater sides of all the false inequalities do not contain the considered weight, the algorithm adjusts the next larger weight. For this example, the weight E is examined first, but it is not adjusted since E does not appear in the greater sides of all the false inequalities #1, #3, #5, and #8. Then, the weight D is considered next and is not adjusted as well due to the same reason.

Next, the weight C is adjusted since it appears in the greater sides of the false inequalities #5 and #8. The increment of a weight is one each time for having a minimal increase of weights. Note that the updated weights have to be compatible with the variable ordering. For this example, when the weight C is increased from 3 to 4, the weights B and A are increased to 5 and 6 as well, respectively, for being compatible with the variable ordering. As a result, the weighted summation of the system of inequalities of s is summarized in Fig. 15(a), where inequality #8 becomes a true inequality.

However, when A, B, C, D, and E are assigned as $(7, 5, 4, 2, 1)$ using the same procedure as shown in Fig. 15(b), some original true inequalities, #4, #7, and #11, become false inequalities though. We call this a *flip* situation. Note that the flip situation could lead the weight assignment procedure to be inefficient or even failed. Then, the algorithm increases the largest weight in the greater side of each inequality, i.e., B is from 5 to 6 and C is from 4 to 5, to make them become true again. Unfortunately, three original false inequalities #1, #3, and #5 are not held again under this new weight assignment $(7, 6, 5, 2, 1)$ as shown in Fig. 16. As a result, the algorithm cannot find a weight assignment satisfying all the inequalities, and s is identified as an undetermined function. In fact, s is TF with the weight assignment $(8, 6, 5, 3, 1)$.

In summary, two key factors make the state-of-the-art [19] fail to identify some TFs.

1) It adjusts weights in a fixed sequence from the smallest weight to the largest weight, which may miss some opportunities of adjusting the most appropriate weight.
2) When encountering a flip situation, it simply increases the largest weight in the greater side of each false inequality, which cannot completely solve the flip situation.

2) Proposed Weight Assignment Method: In this section, we present the proposed weight assignment method. We use a TF p to explain our weight assignment algorithm, where p cannot be identified as a TF by the state-of-the-art as well. Given $p = x_1x_2x_3+x_1x_2x_4+x_1x_2x_5x_6+x_1x_2x_5x_7+x_1x_3x_4+x_1x_3x_5x_6+x_1x_4x_5x_6+x_2x_3x_4+x_2x_3x_5x_6+x_2x_4x_5x_6$ with the variable ordering $x_1 = x_2 > x_3 = x_4 > x_5 > x_6 > x_7$, and its system of irredundant inequalities is as shown in Fig. 17. By replacing

#	Inequality		
1	**A+A = 14**	**>**	**B+B+D = 14** **(false)**
2	A+E = 8	>	C+D = 7
3	**A = 7**	**>**	**B+E = 7** **(false)**
4	B+D = 8	>	A = 7
5	**A+C = 12**	**>**	**B+B = 12** **(false)**
6	C+D = 7	>	B = 6
7	C+D+E = 8	>	A = 7
8	C = 5	>	D+E = 3
9	B+C =11	>	A+E = 8
10	B+B = 12	>	A+D = 9
11	B+B+C = 17	>	A+A = 14

Fig. 16. Weighted summation of the system of irredundant inequalities of s under the weight assignment $(7, 6, 5, 2, 1)$.

#	Inequality		
1	w_1+w_2	>	$w_3+w_4+w_6$
2	w_1	>	$w_5+w_6+w_7$
3	w_4	>	w_5+w_7
4	w_3+w_4	>	$w_2+w_6+w_7$
5	w_3+w_4	>	w_2+w_5
6	w_1	>	w_4+w_7
7	w_3+w_5	>	w_2+w_7
8	w_3+w_6	>	w_2

Fig. 17. System of irredundant inequalities of p.

#	Inequality		
1	A+A	>	B+B+D
2	A	>	C+D+E
3	B	>	C+E
4	B+B	>	A+D+E
5	B+B	>	A+C
6	A	>	B+E
7	B+C	>	A+E
8	B+D	>	A

Fig. 18. System of irredundant inequalities of p with reformatted symbols.

TABLE I
DEFINITIONS OF SYMBOLS $G(w_i)$, $L(w_i)$, AND $d(w_i)$

$G(w_i)$: Times of w_i in the greater side of the false inequalities
$L(w_i)$: Times of w_i in the lesser side of the false inequalities
$d(w_i) = G(w_i) - L(w_i)$

Weight assignment (5, 4, 3, 2, 1)
d(A) = 2, d(B) = 2, d(C) = -3, d(D) = -3, d(E) = -4

#	Inequality			
1	**A+A = 10**	**>**	**B+B+D = 10**	**(false)**
2	**A = 5**	**>**	**C+D+E = 6**	**(false)**
3	**B = 4**	**>**	**C+E = 4**	**(false)**
4	**B+B = 8**	**>**	**A+D+E = 8**	**(false)**
5	**B+B = 8**	**>**	**A+C = 8**	**(false)**
6	**A =5**	**>**	**B+E = 5**	**(false)**
7	B+C = 7	>	A+E = 6	
8	B+D = 6	>	A = 5	

Fig. 19. Weighted summation of the system of irredundant inequalities of p under the weight assignment (5, 4, 3, 2, 1).

the same weight w_i with the same symbol, the reformatted system of irredundant inequalities is as shown in Fig. 18. The initial weight assignment is $A = 5, B = 4, C = 3, D = 2$, and $E = 1$, and the weighted summation of the system of irredundant inequalities of p under the initial assignment is as shown in Fig. 19. In Fig. 19, we see that the inequalities #1–#6 are false inequalities. However, instead of adjusting the weight in a fixed sequence under the VWO as the state-of-the-art [19] did, we present a more comprehensive method to adjust the weights that are called critical weights (CWs). In fact, when a weight increment can turn more false inequalities into true inequalities, this weight is an appropriate weight to be adjusted. Definition 1 defines the CW. Table I lists the definitions of symbols $G(w_i)$, $L(w_i)$, and $d(w_i)$.

Definition 1: The CWs are the weights whose $d(w_i) > 0$.

When there are more than one CW among the weights, we individually adjust the top x of them based on the magnitude of $d(w_i)$ for avoid missing proper assignments, where x can be set by users. Note that the larger x is set, the more TFs will be identified. But the weight assignment procedure needs more iterations. In our weight assignment procedure, we set x as 3 to trade the efficiency off against the effectiveness.

For example, in this function p, $d(A) = G(A) - L(A) = 4 - 2 = 2$, $d(B) = 5 - 3 = 2$, $d(C) = -3$, $d(D) = -3$, and $d(E) = -4$. Hence, the CWs are A and B. By adjusting A, the weight assignment becomes (6, 4, 3, 2, 1); by adjusting B, the weight assignment becomes (6, 5, 3, 2, 1). Note that weight A will be increased as well when B is equal to A after increasing B.

Weight assignment (6, 5, 3, 2, 1)
d(A) = 4, d(B) = -3, d(C) = -1, d(D) = -2, d(E) = -2

#	Inequality			
1	**A+A = 12**	**>**	**B+B+D = 12**	**(false)**
2	**A = 6**	**>**	**C+D+E = 6**	**(false)**
3	B = 5	>	C+E = 4	
4	B+B = 10	>	A+D+E = 9	
5	B+B = 10	>	A+C = 9	
6	**A =6**	**>**	**B+E = 6**	**(false)**
7	B+C = 8	>	A+E = 7	
8	B+D = 7	>	A = 6	

Fig. 20. Weighted summation of the system of irredundant inequalities of p under the weight assignment (6, 5, 3, 2, 1).

Fig. 20 is the weighted summation of the system of irredundant inequalities of p under the weight assignment (6, 5, 3, 2, 1). Three false inequalities are left to be solved.

Next, since $d(A) = 4$, and $d(B)$ to $d(E)$ are all negative values, we increase weight A only in this iteration, and the new weight assignment becomes (7, 5, 3, 2, 1) as shown in Fig. 21(top). When the weight assignment is increased to (8, 6, 3, 2, 1), we have $d(A) = -2$, $d(B) = 2$, $d(C) = 1$, $d(D) = 1$, $d(E) = -1$, and B, C, and D are CWs as shown in Fig. 22. Hence, the weight assignment is increased to (8, 7, 3, 2, 1), (8, 6, 4, 2, 1), and (8, 6, 4, 3, 1), respectively. At last, the correct weight assignment (9, 7, 4, 3, 1) satisfying all the inequalities is obtained by running the same procedure, and the weight assignment algorithm is terminated. The complete weight assignment procedure is summarized in Fig. 21.

Note that two additional conditions for terminating the weight assignment procedure are presented as follows.

1) When there is no CW i.e., each $d(w_i) \leq 0$ in an iteration, the function is identified as a non-TF and the weight assignment procedure is terminated. This is because when each $d(w_i) \leq 0$, the sum of all the weighted summations in the greater side is less than or equal to that in the lesser side. Obviously, this condition violates the linear separability property of TF. Thus, we can confirm that the function is not a TF.
2) When an assigned weight is larger than the theoretically maximum value, the function is identified as an undetermined function and the weight assignment procedure is terminated. This is because the author in [18] has shown an upper bound of maximum weight of an n-input TF, e.g., the maximum weight is 76 for an eight-input TF, and our approach increases the weights from the least positive integer gradually. Hence, when one assigned weight is greater than this upper bound value, the function is an undetermined function.

D. Threshold Value Computation

After having the weight assignment of a TF, we compute its threshold value. According to the linear separability property of a TF, we have known that all the weighted summations in the on-set are definitely greater than that in the off-set. Thus, we can obtain the least threshold value by increasing the largest weighted summation in the off-set by 1.

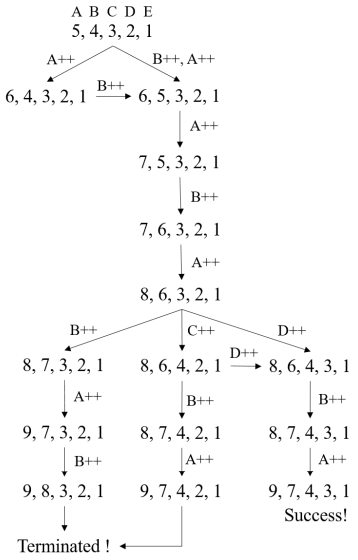


Fig. 21. Process of weight assignments for p.

Weight assignment (8, 6, 3, 2, 1)
d(A) = -2, d(B) = 2, d(C) = 1, d(D) = 1, d(E) = -1

#	Inequality			
1	A+A = 16	>	B+B+D = 14	
2	A = 8	>	C+D+E = 6	
3	B = 6	>	C+E = 4	
4	B+B = 12	>	A+D+E = 11	
5	B+B = 12	>	A+C = 11	
6	A =8	>	B+E = 7	
7	**B+C = 9**	**>**	**A+E = 9**	**(false)**
8	**B+D = 8**	**>**	**A = 8**	**(false)**

Fig. 22. Weighted summation of the system of irredundant inequalities of p under the weight assignment (8, 6, 3, 2, 1).

Greater side		*Lesser side*
$w_1+w_2+w_5+w_7$		$w_3+w_4+w_5+w_6+w_7$
$w_1+w_3+w_4$	$\geq T >$	$w_1+w_3+w_5+w_7$
$w_1+w_3+w_5+w_6$		$w_1+w_2+w_6+w_7$
		$w_1+w_2+w_5$

Fig. 23. List of irredundant weighted summations in the greater side and lesser side of p.

For example, regarding the above mentioned function p, the list of weighted summations for the on-set and off-set cubes is as shown in Fig. 23. The weighted summations in the off-set are 22, 21, 22, and 22, respectively. Hence, the threshold value is computed as $22 + 1 = 23$. As a result, we identify the function p as a TF successfully with the weight-threshold vector $<9, 9, 7, 7, 4, 3, 1; 23>$.

E. Overall Flowchart of TF Identification Algorithm

This section summarizes the overall flowchart of TF identification algorithm as shown in Fig. 24. Given a Boolean function f in ISOP form, the algorithm first checks the unateness of f. If f is not unate, it is identified as a non-TF; otherwise, the algorithm determines the variable ordering based on the modified Chow's parameter. Then, the algorithm generates the system of irredundant inequalities. If there exists an inequality against the variable ordering, f is identified as a non-TF; otherwise, the algorithm gives a weight assignment. If all the inequalities are satisfiable under the assignment, it computes the threshold value and f is identified as a TF; otherwise, the algorithm enters a weight adjustment loop. That is, for each iteration, the algorithm computes whether there exist CWs to be increased. If there exists a CW to be increased, the algorithm adjusts weight about the CW and checks whether all the inequalities are satisfied under the new weight assignment; otherwise, f is identified as a non-TF. Note that the weight adjustment loop also checks whether the largest assigned weight is equal to the theoretically maximum weight value of the function. If it is the case, f is identified as an undetermined function; otherwise, the algorithm runs for the next iteration.

F. Pseudo Code and Time Complexity of TF Identification Algorithm

The pseudo codes of inequality simplification and weight assignment are as shown in Figs. 25–27. For the redundant weighted summation removal algorithm and redundant inequality removal algorithm, given the numbers of weighted summation in the on-set n, and in the off-set f, the time complexity is $O(n^2 + f^2)$ and $O(n * f)$, respectively. The weight assignment procedure is presented as a pseudo code in Fig. 27. Given the user-defined parameter X, the upper bound of theoretically maximum value m, the number of the false inequalities f, and the number of inputs n, the time complexity of weight assignment algorithm is $O(X^m * f * n)$. It seems that this complexity is high. However, the algorithm is practically efficient when considering the following three pruning situations. First, since the CWs are the weights with positive $d(w_i)$, the number of CWs is likely smaller than X in some iterations. Second, when there is no CW in a certain iteration, the procedure of weight assignment will be terminated. Finally, when a valid weight assignment satisfying all the inequalities is obtained, the procedure of weight assignment will be terminated. On the other hand, the required CPU time is strongly related to the magnitude of weight assignment, which determines the number of iterations. By examining all the eight-input TFs, we found that the maximum weight of more than 60% of TFs is smaller than 20, which is far below the upper bound of theoretically maximum value, 76. That means the proposed approach would be terminated earlier for most cases. This also matches our experimental results, 0.08 s for identifying an eight-input TF on average.

IV. Experimental Results

We implemented the proposed algorithm in C++ language. The experiments were conducted on a 2.6-GHz Linux platform (CentOS 6.7). The experiments consist of three parts. The first part is to show the effectiveness of the proposed method in the TF identification for 1 to 8-input functions. The second one is to show the optimality of computed weight assignments and threshold value of the identified TFs. The last experiment is to show the applicability of the proposed approach to the functions with 9–15 inputs.

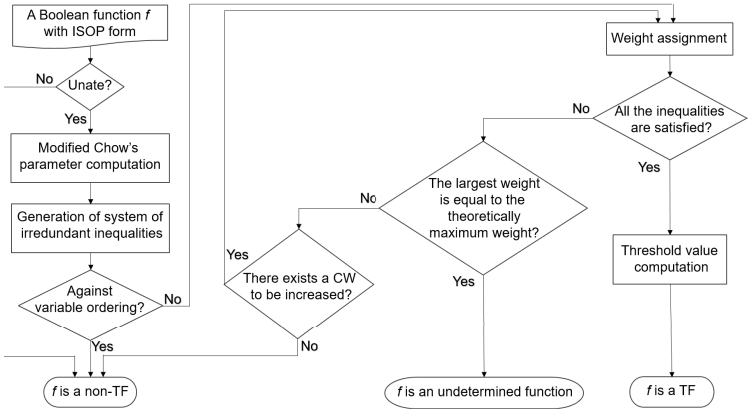


Fig. 24. Overall flowchart of TF identification.

Algorithm Redundant Weighted Summation Removal

Input: Weighted summations in the On-set *WS-On-set* and that in the Off-set *WS-Off-set*
Output: Irredundant_weighted_summations<On-set, Off-set> *IWS*

```
1:  set redundant_list = ∅, IWS.On-set = ∅, IWS.Off-set = ∅
2:  for each weighted summation x_i in WS-On-set
3:      if (x_i is in redundant_list) continue // is x_i redundant?
4:      for each weighted summation x_j in WS-On-set, and j > i
5:          if (x_j is in redundant_list) continue // is x_j redundant?
6:          if (Can_be_Compared (x_i, x_j) == true)
    // can x_i be compared with x_j under the variable ordering?
7:              if (x_i >= x_j) add x_i in redundant_list, then break
    // if x_i is larger than or equal to x_j, x_i is redundant.
8:              else add x_j in redundant_list
9:              endif
10:         endif
11:     endfor
12:     if (x_i is not in redundant_list) add x_i in IWS.On-set
13: endfor
14: initialize(redundant_list)
15: for each weighted summation x_i in WS-Off-set
16:     if (x_i is in redundant_list) continue // is x_i redundant?
17:     for each weighted summation x_j in WS-Off-set, and j > i
18:         if (x_j is in redundant_list) continue // is x_j redundant?
19:         if (Can_be_Compared (x_i, x_j))
    // can x_i be compared with x_j under the variable ordering?
20:             if (x_i <= x_j) add x_i in redundant_list, then break
    // if x_i is smaller than or equal to x_j, x_i is redundant.
21:             else add x_j in redundant_list
22:             endif
23:         endif
24:     endfor
25:     if (x_i is not in redundant_list) add x_i in IWS.Off-set
26: endfor
27: return IWS
```

Fig. 25. Pseudo code of redundant weighted summation removal algorithm.

Algorithm Redundant Inequality Removal

Input: Irredundant_weighted_summations<On-set, Off-set> *IWS*
Output: Irredunant_inequality_system *IIS*

```
1:  set IIS = ∅
2:  for each weighted summation x_i in IWS.On-set
3:      for each weighted summation x_j in IWS.Off-set
4:          if (Can_be_Compared (x_i, x_j))
    // can x_i be compared with x_j under the variable ordering?
5:              if (x_i > x_j) continue
    //the pair of x_i and x_j is redundant.
6:              else goto Termination
    //this condition violates the linear separability property of a TF.
7:              endif
8:          else add pair(x_i, x_j) in IIS
9:          endif
10:     endfor
11: endfor
12: return IIS
13: Termination: the function is a non-TF
```

Fig. 26. Pseudo code of redundant inequality removal algorithm.

A. Effectiveness

As mentioned, to identify a function as a non-TF is easy, but to identify a function as a TF is more challenging. To show the effectiveness of our method in the TF identification, we need to have all the NP-class TFs at hand for the first experiment. However, in [18], only the total numbers of n-input NP-class TFs are available. Furthermore, the set of all n-input NP-class TFs is not available publicly as well. Hence, we first generate the set of all n-input NP-class TFs for $n = 1$ to 8 for our experiments.

Let us introduce our method of generating NP-class TFs. The method assigns the theoretically maximum weight of an n-input function, $\max_n$, obtained from [18] to each weight, and sets $\max_n \times n$ as the threshold value. Then, the algorithm generates a new weight assignment by decreasing the weights or threshold value of 1 iteratively until one of weight values or threshold value is less than 1. The pseudo code of TF generation algorithm is as shown in Fig. 28. Although this approach tried all combinations to find all the TFs, we used some ideas to speed up this process as follows.

1) If the weights and threshold value have a common divisor, for example, $<4, 4, 4, 4; 16>$, the process will skip this combination since we can definitely find the same TF from $<1, 1, 1, 1; 4>$.
2) If the LTG has useless inputs [13], we will skip this combination. A useless input is defined as the input when

Algorithm Weight Assignment

Input: Irredunant_inequality_system *IIS*, the upper bound of theoretically maximum value *upper_bound,* the user-defined parameter X
Output: An LTG *Gate*

```
1:  set queue = ∅, Gate.threshold = ∅, Gate.wa = ∅, false_inequalities = ∅, visit = ∅, satisfaction = ∅
2:  initial_wa = initial_weight_assignment()
    // it assigns a least positive integer to each weight.
3:  queue.push(initial_wa)
4:  while(queue is not empty)
5:      cur_wa = queue.front(), and then queue.pop()
6:      false_inequalities = get_false_inequalities(IIS, cur_wa)
    // it gets the set of false inequalities under current weight assignment.
7:      for each inequality x_i in false_inequalities
8:          initialize(G, L, d) // initialize the arrays of G, L, d
9:          for each w_j
    // it counts the number of w_j in the greater side and lesser side of x_i.
10:             G(w_j) += x_i.greater_side.count(w_j)
11:             L(w_j) += x_i.lesser_side.count(w_j)
12:             d(w_j) += G(w_j) - L(w_j)
13:         endfor
14:     endfor
15:     if (all d(w_i) <=0) goto Termination1
    // this condition violates the linear separability property of a TF.
16:     rank_computation(d)
    // by d(w_i), it determines the rank of each w_i.
17:     for each w_i
18:         if (rank(w_i)<=X && w_i<upper_bound)
19:             new_wa = increment (w_i)
20:             if (visit[new_wa] == false)
21:                 if (check_satisfaction (IIS, cur_wa) == true)
    // does current weight assignment satisfy all the inequalities?
22:                     satisfaction = true
23:                     Gate.wa = cur_wa
24:                     Gate.threshold = threshold_value_computation(IIS, cur_wa)
25:                     return Gate
26:                 else
27:                     queue.push(new_wa)
28:                     visit[new_wa]=true
                    endif
29:             endif
30:         endif
31:     endfor
32: endwhile
33: if (satisfaction==false) goto Termination2
34: Termination1: the function is a non-TF
35: Termination2: the function is an undetermined function
```

Fig. 27. Pseudo code of weight assignment algorithm.

it toggles under all combinations, the output of LTG is still intact. For example, the input with respect to the weight 1 in $<5, 5, 1; 10>$ is a useless input. The reason that we can skip this combination is because the corresponding variable of the useless input will not exist in its ISOP expression. As a result, when an n-input LTG has useless inputs, its ISOP expression will represent a function with fewer inputs, which belongs to the other categories of inputs.

During the iterations, when an LTG with a new weight-threshold vector is generated, we compute the set of CEVs [15] of this LTG, where a set of CEV can uniquely represent a Boolean function. That is, two LTGs with different weight-threshold vectors will be functionally equivalent if and only if

TABLE II
COMPARISON OF THE NUMBER OF IDENTIFIED NP-CLASS TFS AMONG NON-ILP-BASED APPROACHES AND OUR APPROACH

|Input|	[18]	Identified					
		[10]		[19]		Ours	
	|Total TF|	|TF|	%	|TF|	%	|TF|	%
1	1	1	100	1	100	1	100
2	2	2	100	2	100	2	100
3	5	5	100	5	100	5	100
4	17	15	88.2	17	100	17	100
5	92	52	56.5	92	100	92	100
6	994	181	18.2	994	100	994	100
7	28262	573	2.0	22477	79.5	28262	100
8	2700791	1635	0.06	-	-	2700791	100

Algorithm TF_Generation

Input: The upper bound of the maximum value *upper_bound*, an array of *weight*, the index of weight *index*
Output: An *ISOP*

```
1:  if (index is equal to 0)
2:      for variable i from upper_bound to 1
3:          weight[index] = i //assign the index^th weight in an LTG.
4:          TF_Generation (weight[index], weight, index +1)
5:      endfor
6:  else if (index is not equal to the number of input)
7:      for variable i from weight[index -1] to 1
8:          weight[index] = i //assign the index^th weight in an LTG.
9:          TF_Generation (weight[index], weight, index +1)
10:     endfor
11: else // index is equal to the number of input.
12:     initialize(thresholdmax)
    // finish all weight assignments.
13:     for variable i from 0 to index-1
14:         thresholdmax+= weight[i]
15:     endfor
16:     for variable thvalue from thresholdmax to 1
17:         CEV = generateCEV(weight, thvalue)
    // generate the corresponding CEV from the weights and threshold value by [13].
18:         ISOP = generateISOP(CEV)
    // transform CEV into ISOP.
19:         If (ISOP.isNewFunction()) AddtoFunctionPool(ISOP)
20:     endfor
21: endif
```

Fig. 28. Pseudo code of TF generation algorithm.

they have the same set of CEVs. Hence, when a new LTG has a new set of CEVs, its corresponding Boolean function is a new TF. By this method, we can generate all the NP-class TFs with 1–8 inputs in ISOP form, and ensure that each generated TF can be represented by an LTG with positive integers.

In this experiment, we compare our results against the first non-ILP-based approach for TF identification [10] and the state-of-the-art [19] in terms of the numbers of identified TFs with 1–8 inputs. The experimental results are summarized in Table II. Column 1 lists the number of inputs in TFs. Column 2 lists the total numbers of NP-class TFs with 1–8 inputs, which were provided in [18]. Columns 3, 5, and 7 present the numbers of TFs identified by [10], [19], and our approach, respectively. Columns 4, 6, and 8 show the corresponding percentages of the identified TFs in [10] and [19], and our approach, respectively, compared with the total number of TFs.

According to Table II, we observed that [10] can only identify all the TFs within three inputs, and identify 88.2%, 56.5%, 18.2%, 2.0%, and 0.06% of TFs for 4 to 8-input functions, respectively. For the state-of-the-art [19], the TFs within six inputs are completely identified. However, only 79.5% of seven-input TFs are identified, and no result reported for eight-input TFs. In our approach, we identify all the TFs within eight inputs successfully. This is the first non-ILP-based work that reaches this achievement.

B. Optimality

Generally, compared with the ILP-based approach, non-ILP-based approaches can identify a TF more efficiently if they succeed. However, the obtained weights or threshold value are usually not optimal. Hence, in this experiment, we would like to show the obtained weights and threshold value in our approach are the same with the optimal values for 99.4% of cases.

The benchmarks we used in this experiment are all TFs with 1–8 inputs as well. We compare our results with an ILP-based approach [29] in terms of CPU time and percentage of identified TFs that have optimal weights and threshold values. In [29], the optimization models were expressed in the AMPL modeling language [7], and solved by the solver Lp_Solve [5]. The experimental results are summarized in Table III. Column 1 lists the number of inputs. Columns 2 and 4 show the CPU time of [29] and our approach measured in seconds, respectively. Column 3 is the number of LTGs with the optimal weights and threshold value obtained by Zhang *et al.* [29]. Since [29] is an ILP-based approach, which can obtain the optimal results, the numbers in this Column are identical to the total number of TFs. Column 5 shows the CPU time

TABLE III
COMPARISON OF CPU TIME AND THE NUMBER OF OBTAINED OPTIMAL LTGS BETWEEN AN ILP-BASED APPROACH AND OUR APPROACH

|Input|	[29]		Ours			
	CPU(s)	|Opt. LTG|	CPU(s)	Ratio1	|Opt. LTG|	Ratio2
1	<0.01	1	<0.01	-	1	1
2	<0.01	2	<0.01	-	2	1
3	<0.01	5	<0.01	-	5	1
4	2.1	17	<0.01	<0.01	17	1
5	16.3	92	<0.01	<0.01	92	1
6	298	994	<0.01	<0.01	994	1
7	10920	28262	8.78	<0.01	28249	0.9995
8	2826780	2700791	251700	0.08	2695746	0.9981

TABLE IV
IDENTIFICATION OF RANDOMLY GENERATED 100 000 TFS WITH 9–15 INPUTS

|Input|	|Gen. TF|	Ours		
		|TF|	%	CPU(s)
9	100000	100000	100	12.13
10	100000	100000	100	16.24
11	100000	100000	100	23.96
12	100000	100000	100	33.96
13	100000	100000	100	54.14
14	100000	100000	100	95.60
15	100000	100000	100	175.21

ratio (ratio1) of our approach to [29]. Column 6 is the number of TFs having the optimal weights and threshold value by our approach, and the corresponding ratio (ratio2) is shown in the last column.

According to Table III, we observed that the CPU time of our approach is much less than that of ILP-based approach [29] for the 4 to 8-input TFs, indicating that our approach is very efficient. For example, for all the eight-input TFs, [29] needs 2 826 780 s (more than 32 days) to identify them. However, we only need 251 700 s to identify them instead. The CPU time ratio is 0.08.

We also observed that for the eight-input TFs, our approach took more time compared to dealing with the smaller input TFs. The reasons are analyzed as follows.

1) For many eight-input TFs, two CWs are needed to be increased for most iterations in the procedure of weight assignment. Hence, the search space grows significantly.
2) The weight values of many eight-input LTGs are large such that the weight assignment procedure needs more iterations to reach the weights of the LTGs. Note that many of eight-input TFs cannot be identified by other heuristics according to Table II.

Next, we discuss the optimality about the results of our approach. The LTGs obtained by the ILP-based approach [29] are called the optimal LTGs due to the optimal weights and threshold value. Hence, we count the number of LTGs that are optimal LTGs obtained by our approach. The results are shown in Table III and demonstrate that more than 99.4% of obtained LTGs are optimal LTGs for all the TFs with 1–8 inputs by our approach. These results indicate that our approach is effective in identifying TFs with high qualities.

The reason for missing the optimal weights and threshold value for a tiny number of cases of 7, eight-input TFs is as follows. Our weight assignment strategy is based on a breadth first search algorithm. Once the strategy obtains a weight assignment satisfying the system of inequalities, it will be terminated instead of continuing the process of weight assignment for traversing the whole solution space. As a result, the proposed identification algorithm could miss the optimal weight assignments and threshold value when a suboptimal solution has been found already. If we want to obtain the optimal weight assignment and threshold value for all the TFs, the algorithm would be changed to search for all the valid solutions and then pick up the optimal one.

C. Applicability

To show that our approach is also effective and efficient for identifying TFs with more than eight inputs, we conduct this experiment for identifying TFs with 9–15 inputs. Since the number of all the TFs with more than eight inputs is too large to be enumerated, we randomly generate 100 000 TFs for an input category to evaluate the effectiveness and efficiency of our approach. From another viewpoint, most synthesized threshold circuits have no LTG with more than 15 inputs due to physical implementation concern. Thus, we decide to identify TFs with 9–15 inputs to show the applicability of our approach.

The results of this experiment are summarized in Table IV. According to Table IV, our approach is able to identify all the randomly generated 100 000 TFs with 9–15 inputs, and it spent less than 0.002 s for identifying a TF on average. This result demonstrates the efficiency of our approach. Let us discuss the growth of CPU time with respect to the number of inputs. Since the TFs are generated randomly, the value of each weight of an LTG cannot be predetermined. That means the scales of weights in different inputs might be completely different. Due to this reason, the growth of CPU time is various for different input categories as expected.

V. Conclusion

In this paper, we propose an efficient and effective approach for TF identification. We first significantly decrease the number of inequalities by removing redundancies. Then, we present a comprehensive weight assignment procedure. As the experimental results demonstrated, our approach is the first non-ILP-based approach that is able to identify all the NP-class TFs with 1–8 inputs. Furthermore, more than 99.4% of resultant LTGs of TFs are optimal LTGs. Last, we showed the applicability of our approach for identifying TFs with 9–15 inputs in a reasonable CPU time. Our achievements advance the theoretical research in threshold logic and could improve the synthesis and optimization of threshold logic circuits.

References

[1] C. Augustine *et al.*, "Low-power functionality enhanced computation architecture using spin-based devices," in *Proc. Int. Symp. Nanoscale Archit.*, 2011, pp. 129–136.

[2] M. J. Avedillo, J. M. Quintana, H. Pettenghi, P. M. Kelly, and C. J. Thompson, "Multi-threshold threshold logic circuit design using resonant tunnelling devices," *Electron. Lett.*, vol. 39, no. 21, pp. 1502–1504, Oct. 2003.

[3] I. A. Basheerg and M. Hajmeer, "Artificial neural networks: Fundamentals, computing, design, and application," *J. Microbiol. Methods*, vol. 43, no. 1, pp. 3–31, Dec. 2000.

[4] V. Beiu, J. M. Quintana, and M. J. Avedillo, "VLSI implementations of threshold logic—A comprehensive survey," *IEEE Trans. Neural Netw.*, vol. 14, no. 5, pp. 1217–1243, Sep. 2003.

[5] M. Berkelaar, K. Eikland, and P. Notebaert. *LP_Solve 5.5, Open Source (Mixed-Integer) Linear Programming System. Release 5.5*. Accessed: Aug. 2018. [Online]. Available: http://lpsolve.sourceforge.net/5.5/

[6] Y.-C. Chen, R. Wang, and Y.-P. Chang, "Fast synthesis of threshold logic networks with optimization," in *Proc. Asia South Pac. Design Autom. Conf.*, 2016, pp. 486–491.

[7] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, 2nd ed. Danvers, MA, USA: Boyd and Fraser, 2002.

[8] M. J. Ghazala, "Irredundant disjunctive and conjunctive forms of a Boolean function," *IBM J. Res. Develop.*, vol. 1, no. 2, pp. 171–176, Apr. 1957.

[9] D. Goldharber-Gordon, M. S. Montemerlo, J. C. Love, G. J. Opiteck, and J. C. Ellenbogen, "Overview of nanoelectronic devices," *Proc. IEEE*, vol. 85, no. 4, pp. 521–540, Apr. 1997.

[10] T. Gowda, S. Vrudhula, and G. Konjevod, "A non-ILP based threshold logic synthesis methodology," in *Proc. IWLS*, 2007, pp. 222–229.

[11] T. Gowda, S. Vrudhula, N. Kulkarni, and K. Berezowski, "Identification of threshold functions and synthesis of threshold networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 5, pp. 665–677, May 2011.

[12] G. B. Huang *et al.*, "Can threshold networks be trained directly?" *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, no. 3, pp. 187–191, Mar. 2006.

[13] P.-Y. Kuo, C.-Y. Wang, and C.-Y. Huang, "On rewiring and simplification for canonicity in threshold logic circuits," in *Proc. ICCAD*, 2011, pp. 396–403.

[14] Y.-A. Lai, C.-C. Lin, C.-C. Wu, Y.-C. Chen, and C.-Y. Wang, "Efficient synthesis of approximate threshold logic circuits with an error rate guarantee," in *Proc. DATE*, 2018, pp. 773–778.

[15] C.-C. Lin, C.-Y. Wang, Y.-C. Chen, and C.-Y. Huang, "Rewiring for threshold logic circuit minimization," in *Proc. DATE*, 2014, pp. 1–6.

[16] C.-C. Lin, C.-W. Huang, C.-Y. Wang, and Y.-C. Chen, "In&out: Restructuring for threshold logic network optimization," in *Proc. ISQED*, 2017, pp. 413–418.

[17] S. Minato, "Fast generation of prime-irredundant covers from binary decision diagrams," *IEICE Trans. Fundam.*, vol. E76-A, no. 6, pp. 967–973, 1993.

[18] S. Muroga, *Threshold Logic and Its Applications*. New York, NY, USA: Wiley, 1971.

[19] A. Neutzling, M. G. A. Martins, V. Callegaro, A. I Reis, and R. P. Ribas, "A simple and effective heuristic method for threshold logic identification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 5, pp. 1023–1036, May 2018.

[20] V. A. Mardiris, G. C. Sirakoulis, and I. G. Karafyllidis, "Automated design architecture for 1-D cellular automata using quantum cellular automata," *IEEE Trans. Comput.*, vol. 64, no. 9, pp. 2476–2489, Sep. 2015.

[21] S. R. Petrick, "A direct determination of the irredundant forms of a Boolean function from a set of prime implicants," A. F. Cambridge Res. Center, Bedford, MA, USA, Rep. AFCRC-TR-56-110, 1956.

[22] V. Saripalli, L. Liu, S. Datta, and V. Narayanan, "Energy-delay performance of nanoscale transistors exhibiting single electron behavior and associated logic circuits," *J. Low Power Electron.*, vol. 6, no. 3, pp. 415–428, 2010.

[23] C.-K. Tsai, C.-Y. Wang, C.-Y. Huang, and Y.-C. Chen, "Sensitization criterion for threshold logic circuits and its application," in *Proc. ICCAD*, 2013, pp. 226–233.

[24] J. Schlachter, V. Camus, and C. Enz, "Near/sub-threshold circuits and approximate computing: The perfect combination for ultra-low-power systems," in *Proc. ISVLSI*, 2015, pp. 476–480.

[25] R. O. Winder, "Single stage threshold logic," in *Proc. Switch. Circuit Theory Logical Design*, 1961, pp. 321–332.

[26] R. O. Winder, "Threshold logic," Ph.D. dissertation, Dept. Math., Princeton Univ., Princeton, NJ, USA, 1962.

[27] R. O. Winder, "Enumeration of seven-argument threshold functions," *IEEE Trans. Electron. Comput.*, vol. EC-14, no. 3, pp. 315–325, Jun. 1965.

[28] W. V. Quine, "The problem of simplifying truth functions," *Amer. Math. Monthly*, vol. 59, no. 8, pp. 521–531, 1952.

[29] R. Zhang, P. Gupta, L. Zhong, and N. K. Jha, "Threshold network synthesis and optimization and its application to nanotechnologies," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 24, no. 1, pp. 107–118, Jan. 2005.

Chin-Heng Liu received the B.S. degree from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2016, where he is currently pursuing the M.S. degree.

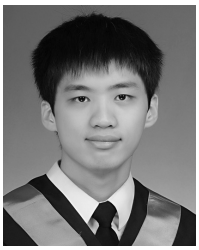
His current research interests include threshold logic synthesis, optimization, verification for very large-scale integration designs, and automation for emerging technologies.

Chia-Chun Lin received the B.S. and M.S. degrees from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2011 and 2013, respectively, where he is currently pursuing the Ph.D. degree.

His current research interests include logic synthesis, optimization, verification for very large-scale integration designs, and automation for emerging technologies.

Yung-Chih Chen received the B.S., M.S., and Ph.D. degrees from the Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, in 2003, 2005, and 2011, respectively.

He is currently an Assistant Professor with the Department of Computer Science and Engineering, Yuan Ze University, Taoyuan, Taiwan. His current research interests include logic synthesis, design verification, and design automation for emerging technologies.



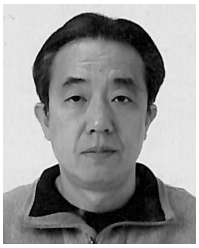
Chia-Cheng Wu received the B.S. degree from the Department of Double Specialty Program of Management and Technology, National Tsing Hua University, Hsinchu, Taiwan, in 2015, where he is currently pursuing the Ph.D. degree with the Department of Computer Science.

His current research interests include diagnosis and logic synthesis for emerging technologies.

Chun-Yao Wang (M'03) received the B.S. degree from the Department of Electronics Engineering, National Taipei University of Technology, Taipei, Taiwan, in 1994, and the Ph.D. degree from the Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, in 2002.

Since 2003, he has been an Assistant Professor with the Department of Computer Science, National Tsing Hua University, Hsinchu, where he is currently a Distinguished Professor. His current research interests include logic synthesis, optimization, and verification for very large-scale integrated/system-on-chip designs and emerging technologies. He has published over 60 technical papers in the above areas and is a named inventor in nine patents.

Dr. Wang was nominated for the Best Paper Award in the 2009 IEEE Asia and South Pacific Design Automation Conference and the 2010 IEEE/ACM Design Automation Conference for two of his research results.



Shigeru Yamashita (SM'13) received the B.E., M.E., and Ph.D. degrees in information science from Kyoto University, Kyoto, Japan, in 1993, 1995, and 2001, respectively.

He is a Professor with the Department of Computer Science, College of Information Science and Engineering, Ritsumeikan University, Kyoto. He has been a Visiting Professor with the National Institute of Informatics since 2012. His current research interests include new types of computation and logic synthesis for them.

Dr. Yamashita was a recipient of the 2000 IEEE Circuits and Systems Society Transactions on Computer-Aided Design of Integrated Circuits and Systems Best Paper Award, the SASIMI 2010 Best Paper Award, the 2010 IPSJ Yamashita SIG Research Award, and the 2010 Marubun Academic Achievement Award of Marubun Research Promotion Foundation. He is a Senior Member of IEICE, and a member of ACM and IPSJ.